PHP Hypertext Preprocessor

A brief survey

Stefano Fontanelli stefano.fontanelli@sssup.it

January 16, 2009

PHP: what is it?

1. Acronym for PHP Hypertext Preprocessor
2. it is a scripting language,
3. it is originally designed for producing dynamic web pages,
4. it has evolved to include a command line interface capability,
5. it also can be used in standalone graphical applications.

PHP availability

1. It is Open Source and free to use,
2. many OS support it,
3. several modules and extensions are available
   for example the extension to connect with MySQL dbms.

PHP for web development

1. It runs on a web server,
2. can be embedded into HTML,
3. usually web framework are used to help development.

An example: hello.php

```
<html>
 <head>
  <title>PHP Hello world!</title>
 </head>
 <body>
  <p><?php echo "Hello World!"; ?></p>
 </body>
</html>
```

PHP syntax: Delimiters

1. PHP only parses code within its delimiters,
2. the purpose of these delimiters is to separate PHP code from non-PHP code,
3. everything outside the delimiters is ignored by the parser and is passed through as output,
4. the most common delimiters are `<?php` and `?>`.

The *Hello World* code example is:

```
<?php
echo "Hello World!";
?>
```

The output is:

```
Hello World!
```

PHP syntax: Comments

PHP supports 'C', 'C++' and Unix shell-style (Perl style) comments.

For example:

```
<?php
echo 'This is a test'; // A one-line c++ style comment

/*
 * This is a multi line comment
 */

echo 'One Final Test'; # A one-line shell-style comment
?>
```

PHP syntax: Constants

1. a constant is an identifier (name) for a constant value,
2. a valid constant name starts with a letter or underscore, followed by any number of letters, numbers, or underscores,
3. a constant is case-sensitive by default,
4. by convention, constant identifiers are always uppercase,
5. you can access constants anywhere in your script without regard to scope.

An example:

```
<?php
define('MYCOSTANT', "Hello world!");
echo MYCONSTANT;
?>
```

The output is:

```
Hello World!
```

PHP syntax: Data types and variables

PHP supports eight primitive types:

1. four scalar types (boolean, integer, float, string),
2. two compound types (array, object),
3. two special types (resource, NULL).

Type does not need to be specified in advance.

1. Variables are represented by a $ followed by its name,
2. a valid variable name follows the same rules of constants,
3. the variable name is case-sensitive.

An example:

```
<?php
$myString = "Hello world!";
echo $myString;
?>
```

The output is:

```
Hello World!
```

PHP syntax: Data types and variables (2)

Examples:

```
<?php
$myBoolean = True;
$myBoolean = False;

$myInteger = 123;
$myInteger = -123;
$myInteger = 0123;
$myInteger = 0x1A;

$myFloat = 1.23;
$myFloat = 1.0e3;
$myFloat = 1E-1;

$myString = 'This string will not be escaped';
$myString = "This string will be escaped";
$myString = "I'll be back!";
?>
```

PHP syntax: Data types and variables (3)

Examples:

```
<?php
$myArray = array("mykey" => "myvalue", 1 => 300);
echo $myArray["mykey"]; // Output:  myvalue
echo $myArray[1]; // Output:  300

$myArray[] = 200;
echo $myArray[2]; // Output:  200

$myArray["newkey"] = "newvalue";
echo $myArray["newkey"]; // Output:  newvalue

unset($myArray["newkey"]); // remove the element
unset($myArray); // delete the whole array
$myArray[] = array(); // create an empty array
?>
```

PHP syntax: Classes and Objects

A simple example:

```
<?php
class MyClass {

 private $myVar = 0;

 public myFunction() {
  echo "\$myVar value is $myVar";
 }

}

$myObj = new myClass();
$myObj->myFunction(); // Output:  $myVar value is 0
?>
```

PHP syntax: Arithmetic and Assignment Operators

Examples:

```
<?php
$a = 0;
$a += 5; // $a = 5
$echo -$a; // print:  -5
$b = $a + 5; // $b = 10
$c = $b - $a; // $c = 5
$d = $a * $c; // $d = 25
$e = $b / $a; // $e = 2
$f = $b % $a; // $e = 0
$b = ++$a; // $b = 6, $a = 6
$c = $b++; // $c = 6, $b = 7
$b = --$a; // $b = 5, $a = 5
$c = $b--; // $c = 5, $b = 4
$s = "Hello " .  "World";
$s .= "!";
echo $s; //Output:  Hello World!

?>
```

PHP syntax: Comparison Operators (1)

Examples

1. `$a == $b` return `TRUE` if `$a` is equal to `$b`;
2. `$a === $b` return `TRUE` if `$a` is equal to `$b` and they are of the same type;
3. `$a != $b` return `TRUE` if `$a` is not equal to `$b`
4. `$a !== $b` return `TRUE` if `$a` is not equal to `$b` or they are not of the same type;
5. `$a < $b` return `TRUE` if `$a` is strictly less than `$b`;
6. `$a > $b` return `TRUE` if `$a` is strictly greater than `$b`;
7. `$a <= $b` return `TRUE` if `$a` is less than or equal to `$b`;
8. `$a <= $b` return `TRUE` if `$a` is greater than or equal to `$b`;

Warning!

1. it is possible the comparison between two different data types,
2. the parser will perform an implicit cast (see the manual).

PHP syntax: Comparison Operators (2)

An examples of implicit casts during the comparisons:

```
<?php
$a = "This is a string";
if (0 == $a) {
 echo "\$a is equal to 0.";
} else {
 echo "\$a is not equal to 0.";
}
$b = "The 4th edition of IMIT."
if (4 == $b) {
 echo "\$b is equal to 4.";
} else {
 echo "\$b is not equal to 4.";
}
?>
```

The output is: `$a is equal to 0. $b is equal to 4.`

The Comparison with Various Types table:
http://www.php.net/manual/en/language.operators.comparison.php

PHP syntax: Logical Operators

Examples

1. `!$a` return `TRUE` if `$a` is not `TRUE`;
2. `$a && $b` return `TRUE` if both `$a` and `$b` are `TRUE`;
3. `$a and $b` return `TRUE` if both `$a` and `$b` are `TRUE`;
4. `$a || $b` return `TRUE` if either `$a` or `$b` are `TRUE`;
5. `$a or $b` return `TRUE` if either `$a` or `$b` are `TRUE`;
6. `$a xor $b` return `TRUE` if either `$a` or `$b` are `TRUE`, but not both.

The precedence of the operators is not the same!
See Operator Precedence of the manual.

PHP syntax: Array Operators

Examples

1. `$a + $b`
   union of `$a` and `$b`;
2. `$a == $b`
   return `TRUE` if `$a` and `$b` have the same key/value pairs;
3. `$a === $b`
   return `TRUE` if `$a` and `$b` have the same key/value pairs
   in the same order and of the same types;
4. `$a != $b`
   return `TRUE` if `$a` is not equal to `$b`;
5. `$a !== $b`
   return `TRUE` if `$a` is not identical to `$b`;

PHP syntax: if/elseif/else

```
<?php
if ($a > $b) {
 echo "\$a is bigger than \$b.";
} elseif ($a == $b) {
 echo "\$a is equal to \$b.";
} else {
 echo "\$a is smaller than \$b.";
}
?>
```

An HTML example:

```
<?php if ($a > $b) { ?>
 <p>a is bigger than b.</p>
<?php } else { ?>
 <p>a is not bigger than b.</p>
<?php } ?>
```

PHP syntax: switch

```
<?php
switch ($i) {
 case 0:
  echo "i equals 0";
  break;
 case 1:
  echo "i equals 1";
  break;
 case 2:
  echo "i equals 2";
  break;
 default:
  echo "i is not equal to 0, 1 or 2";
}
?>
```

PHP syntax: while/do-while

The differences between `while` and `do-while`:

```
<?php

 $i = 0;
 while ($i < 10) {
 // print numbers from 0 to 9
  echo $i++;
 }

 // now $i = 10

 do {
  echo $i++; // print 10
 } while ($i < 10);
 // now $i = 11

?>
```

PHP syntax: for/foreach

```php
<?php
 for ($i = 0; $i < 10; $i++) {
  // print numbers from 0 to 9
  echo $i++;
 }

 $myArray = array(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11);

 foreach ($myArray as $value) {
  if (!($value % 3)) continue;
  if ($value == 10) break;
  echo $value; // print 0, 1, 2, 4, 5, 7, 8
 }
?>
```

PHP syntax: include/require (1)

1. the `include()` and `require()` statements include and evaluate the specified file;
2. the two constructs are identical in every way except how they handle failure;
3. use `require()` if you want a missing file to halt processing of the page.

File: ./includes/functions.php

```
<?php function helloWorld() { echo "Hello World!"; } ?>
```

File: ./hello.php

```
<?php
 include("./functions.php"); // print a warning
 helloWorld(); // print nothing
 include("./includes/functions.php");
 helloWorld(); // print:  Hello World!
?>
```

PHP syntax: include/require (2)

File: ./includes/functions.php

```
<?php function helloWorld() { echo "Hello World!"; } ?>
```

File: ./hello.php

```
<?php
 require("./functions.php"); // FATAL ERROR
 echo "String will never be printed.";
?>
```

PHP has also `include_once()` and `require_once()` statements:

1. they are similar to `include()` and `require()` statements;
2. but if the code from a file has already been included, it will not be included again.

PHP syntax: User-defined functions (1)

A function may be defined using syntax such as the following:

```
<?php
 function myFunction($arg1, $arg2, ..., $argN) {
  echo "An example of user-defined function.";
  return $val;
 }
?>
```

A valid function name starts with a letter or underscore, followed by any number of letters, numbers, or underscores.

It is possible to call recursive functions in PHP.

PHP syntax: User-defined functions (2)

You can pass arguments by reference, and set default arguments values:

```
<?php
 function myFunction(&$value) {
  $value *= 100;
 }
 function myFunction2($arg1, $arg2 = "default") {
  echo $arg1 .  " - " .  $arg2;
 }
 $value = 5;
 myFunction($value);
 echo $value; // print:  500
 myFunction2($value); // print: 500 - default
?>
```

PHP syntax: User-defined functions (3)

Values (any type) are returned by using the optional return statement:

```
<?php
 function myFunction($value) {
  return $value *= 100;
 }
 function &myFunction2() {
  $ref = new MyObject();
  return $ref;
 }
 $myRef &= myFunction2();
?>
```

PHP reference: predefined variables

PHP provides a large number of predefined variables to all scripts:

1. `$_SERVER` an associative array with server and execution environment information;
2. `$_GET` an associative array with HTTP GET variables;
3. `$_POST` an associative array with HTTP POST variables;
4. `$_FILES` an associative array with HTTP File Upload variables;
5. `$_SESSION` an associative array with session variables;
6. `$_COOKIE` an associative array with HTTP cookies;
7. `$argc` the number of arguments passed to script;
8. `$argv` array of arguments passed to script.

PHP reference: handling file uploads (1)

PHP can receive file uploads from any RFC-1867 compliant browser:

```
<!-- The data encoding type, enctype,
     MUST be specified as below -->
<form enctype="multipart/form-data" action="myscripts.php"
      method="POST">
<!-- MAX_FILE_SIZE must precede the file input field -->
<input type="hidden" name="MAX_FILE_SIZE" value="30000" />
<!-- Name of input element determines name
     in $_FILES array -->
Send file 1:  <input name="userfile[]" type="file" />
Send file 2:  <input name="userfile[]" type="file" />
<input type="submit" value="Send File" />
</form>
```

PHP reference: handling file uploads (2)

```
<?php
foreach ($_FILES["userfile"]["error"] as $key => $error) {
  if ($error == UPLOAD_ERR_OK) {
    $tmp_name = $_FILES["userfile"]["tmp_name"][$key];
    $name = $_FILES["userfile"]["name"][$key];
    if (move_uploaded_file($tmp_name, "data/$name")){
      echo "File is valid, and was successfully uploaded.";
    } else {
      echo "File was not successfully uploaded.";
    }
  }
}
?>
```

Exercise

1. install WAMP server on your PC (http://www.wampserver.com);
2. develop a simple web application.

The application is composed of two scripts:

1. the first prints HTML form with
   3 input text field (name, surname, email), 1 input file field
   and a submit button which send the request to second script;
2. the second script receives variables then
   it prints the text variables and saves on disk the file.

References

PHP Manual (English HTML version): http://www.php.net